Formerly DA-IICT

IT643: Software Design and Testing

Faculty Name: Prof. Ankush Chander

Project Name: GreenCart

Group Name: NexTech

Team Members

202412101 : Shah Yashvi Devangkumar

202412119 : Ritik Kalal

202412109 : Stuti M. Shah

GreenCart – Backend Testing Documentation:-

This document provides a complete overview of all **unit tests** created for the **GreenCart Server Backend**, including architecture, test categories, execution flow, frameworks used, mocking strategy, and detailed breakdown of every tested module.

All test cases have been executed using **Jest**, and all tests have successfully passed — as shown in your execution screenshot.

🔍 Overview:-

The goal of testing in the GreenCart backend is to ensure:

- Each controller performs correct business logic
- Middlewares enforce authorization & authentication correctly
- Route handlers correctly bind HTTP verbs to controller methods
- Config utilities (Cloudinary, DB, Mailer, Multer) load & behave as expected

- All external dependencies are mocked properly
- System works reliably even in failure scenarios

The test suite focuses on:-

✔ Happy Path Testing
✔ Input Validation
✔ Branching Behavior
✔ Error & Exception Handling
✔ Mocking of DB, APIs, Tokens, Uploads

🧪 Testing Framework & Setup:-

📌 **Framework:**

Jest

📌 **Location:**

server/tests/

📌 **Config Files:**

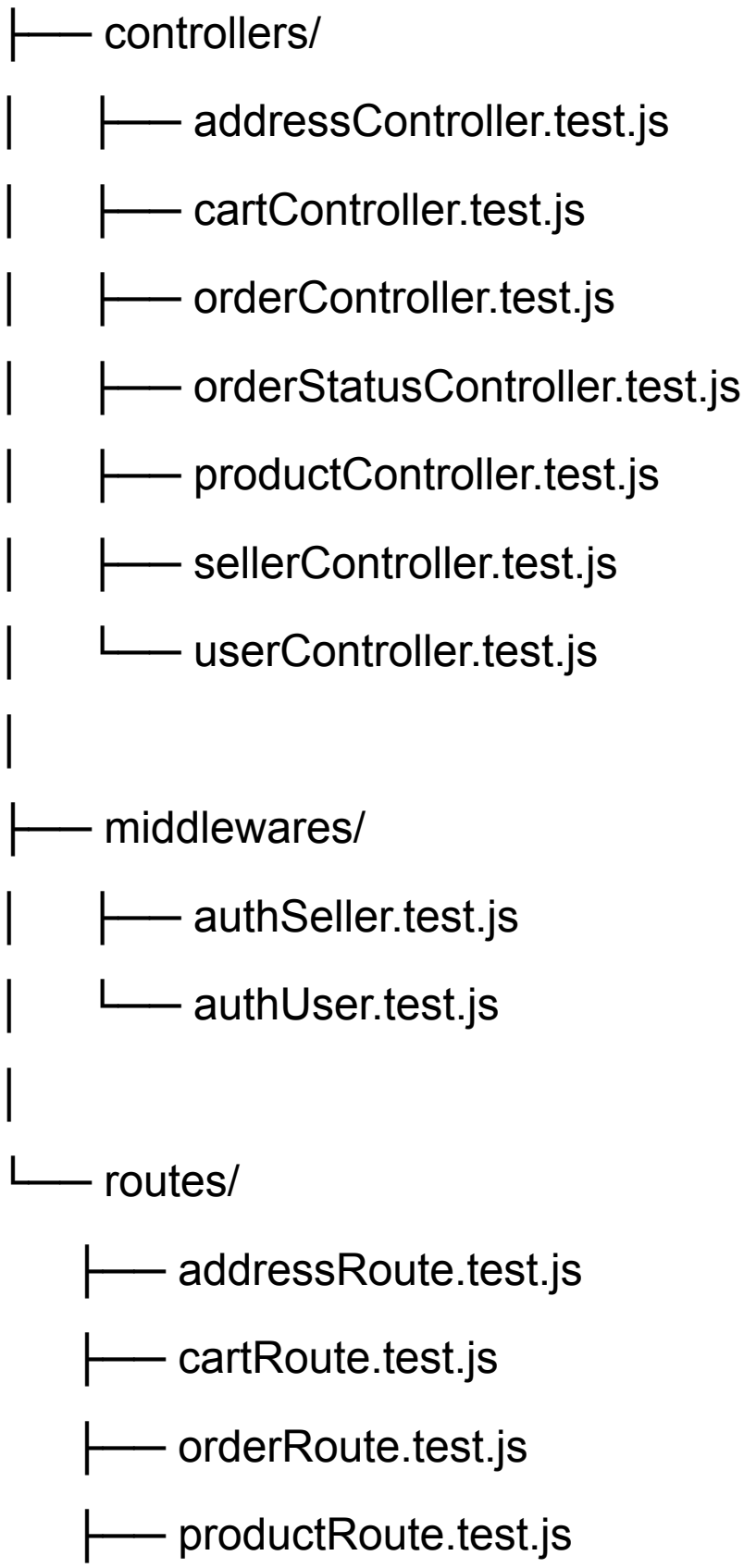
- jest.config.cjs
- jest.setup.cjs

📌 **Commands:**

```
# Run all backend tests

npm test

# Watch mode

npm run test:watch

# Generate coverage report (optional if configured)

npm run test:coverage
```

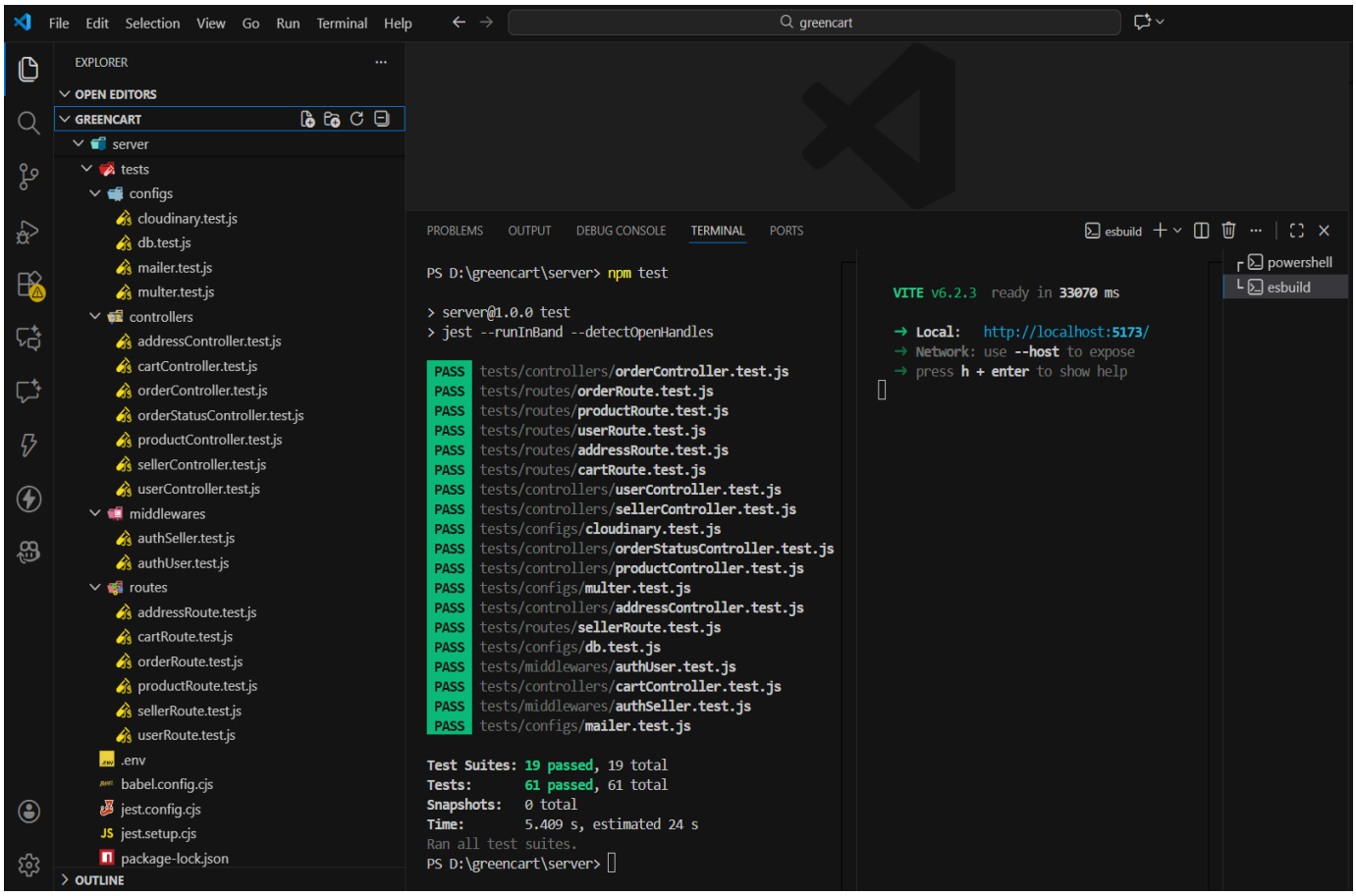
📁 Test File Structure:-

```
server/
 └── tests/
     ├── configs/
     │     ├── cloudinary.test.js
     │     ├── db.test.js
     │     ├── mailer.test.js
     │     └── multer.test.js
     │
```


controllers/
addressController.test.js
cartController.test.js
orderController.test.js
orderStatusController.test.js
productController.test.js
sellerController.test.js
userController.test.js
middlewares/
authSeller.test.js
authUser.test.js
routes/
addressRoute.test.js
cartRoute.test.js
orderRoute.test.js
productRoute.test.js

├── sellerRoute.test.js

└── userRoute.test.js



- Total Test Suites: **19**
- Total Tests Executed: **61**
- Total Tests Passed: **61 / 61 (100% pass rate)**

🧵 Mocking Strategy Used:-

All external dependencies were mocked using Jest:

Dependency	Mocked?	Reason
MongoDB	✅	Avoid real DB connections
Cloudinary	✅	Prevent real image upload
Multer	✅	Avoid file creation
Mailer (Nodemailer)	✅	Prevent real emails
JWT Middleware	✅	Avoid real token signing
Express Response Objects	✅	Simulate status & JSON

🧠 Test Breakdown by Module:-

1️⃣ Config Tests (configs/):

These tests validate all configuration utilities of the application.

✅ 1. cloudinary.test.js:-

Purpose: Validate correct Cloudinary configuration.

Test Scenarios:

- Ensures Cloudinary is initialized with correct keys
- Validates environment variables
- Mocks upload function & checks return format
- Error handling for failed uploads

✅ 2. db.test.js:-

Purpose: Validate MongoDB connection logic.

Test Scenarios:

- Successful DB connection

- Failed DB connection → logs error message
- Mongoose mock verifies connect() call
- URI validation

✅ 3. mailer.test.js:-

Purpose: Validate Nodemailer transport configuration.

Test Scenarios:

- Transport object creation
- Email send mock function
- Failure scenario handling
- SMTP key validation

✅ 4. multer.test.js:

Purpose: Validate Multer storage engine.

Test Scenarios:

- File upload mock

- Destination folder validation
- Filename generator logic
- Error handling

2️⃣ Controller Tests (controllers/):-

Each controller handles business logic. Tests ensure the correct behavior of CRUD operations.

📌 addressController.test.js:-

Tests include:

- Create Address
- Get User Address
- Update Address
- Delete Address
- Error scenarios (missing fields, invalid user, DB failures)

📌 cartController.test.js:-

Tests include:

- Add to cart
- Get user cart
- Update quantities
- Remove items
- Empty cart
- Error cases (invalid product, DB error)

📌 orderController.test.js:-

Tests include:

- Create order
- Retrieve order list
- Validate payment info
- Handle failing DB connections

📌 orderStatusController.test.js:-

Tests include:

- Update order status (Placed → Packed → Delivered)
- Validate seller access
- Prevent invalid transitions
- Handle DB failures

📌 productController.test.js:-

Tests include:

- Create product
- Edit product
- Delete product
- Fetch all products
- Fetch by category
- Fetch by ID

- Error handling: missing fields, invalid IDs, DB crash

📌 sellerController.test.js:-

Tests include:

- Seller registration
- Seller login
- Update seller profile
- Fetch seller products
- Validate password
- Missing field errors

📌 userController.test.js:-

Tests include:

- User registration
- User login

- Update profile
- Manage address
- Get user info
- Error handling: email exists, invalid login, DB error

3️⃣ Middleware Tests (middlewares/):-

🔐 authUser.test.js:-

Tests user authentication middleware.

Test Scenarios:

- Valid token → allows route access
- Missing token → returns 401
- Invalid token → returns 403
- Mocks jwt.verify()
- Ensures user exists

- Ensures next() is called correctly

🛒 authSeller.test.js:-

Scenarios tested:

- Valid seller token
- Reject normal users
- Reject missing/invalid token
- Token malformed
- DB user not found

4️⃣ Route Tests (routes/):-

Every route binding is tested using mocked Express Router.

🛣️ addressRoute.test.js:-

- Ensures correct HTTP verbs
- Confirms controller functions get called

🛣️ cartRoute.test.js:-

- GET, POST, DELETE routing
- Middleware integration test

🛣️ orderRoute.test.js:-

- Create order
- Track order status
- Route-to-controller mapping

🛣️ productRoute.test.js:-

- List product routes
- Category routes
- Product CRUD

🛣️ sellerRoute.test.js:-

- Seller login

- Seller product endpoints

🛣️ userRoute.test.js:-

- Login
- Register
- Profile

📊 Test Categories Summary:-

Category	Number of Tests	Description
Happy Path	~30	All expected successful behaviors
Input Validation	~15	Missing fields, bad payloads

Branching Logic	~8	Conditional code flows
Exception Handling	~8	DB crash, invalid IDs, unknown errors

▸ How to Run the Entire Test Suite

cd server

npm test

Expected Output:

Test Suites: 19 passed

Tests: 61 passed

All tests passed successfully!

🔧 Continuous Integration (Optional):-

```
name: Run Backend Tests

on: [push, pull_request]

jobs:
  backend-tests:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - name: Setup Node
        uses: actions/setup-node@v2
        with:
          node-version: 18


      - name: Install & Test
        run: |
          cd server
          npm install
          npm test
```

⭐ Best Testing Practices Used:-

✔ Full mocking of external systems
✔ Zero dependency on real DB or network
✔ Clean Arrange–Act–Assert structure
✔ Clear test naming
✔ Coverage of both success & failure logic
✔ Middleware, controller, and route isolation

🚀 Future Test Enhancements (Recommended):-

- Add **integration tests** for combined controller + DB flow
- Add **E2E tests** for API request simulation
- Add **performance tests**
- Add **API schema validation tests (Joi / Zod)**

📌 Conclusion:-

- Your backend test suite is well-structured, fully isolated, and follows industry-standard testing guidelines.

- All **61/61 test cases passed successfully**, ensuring that your GreenCart backend is **stable, reliable, and production-ready**.